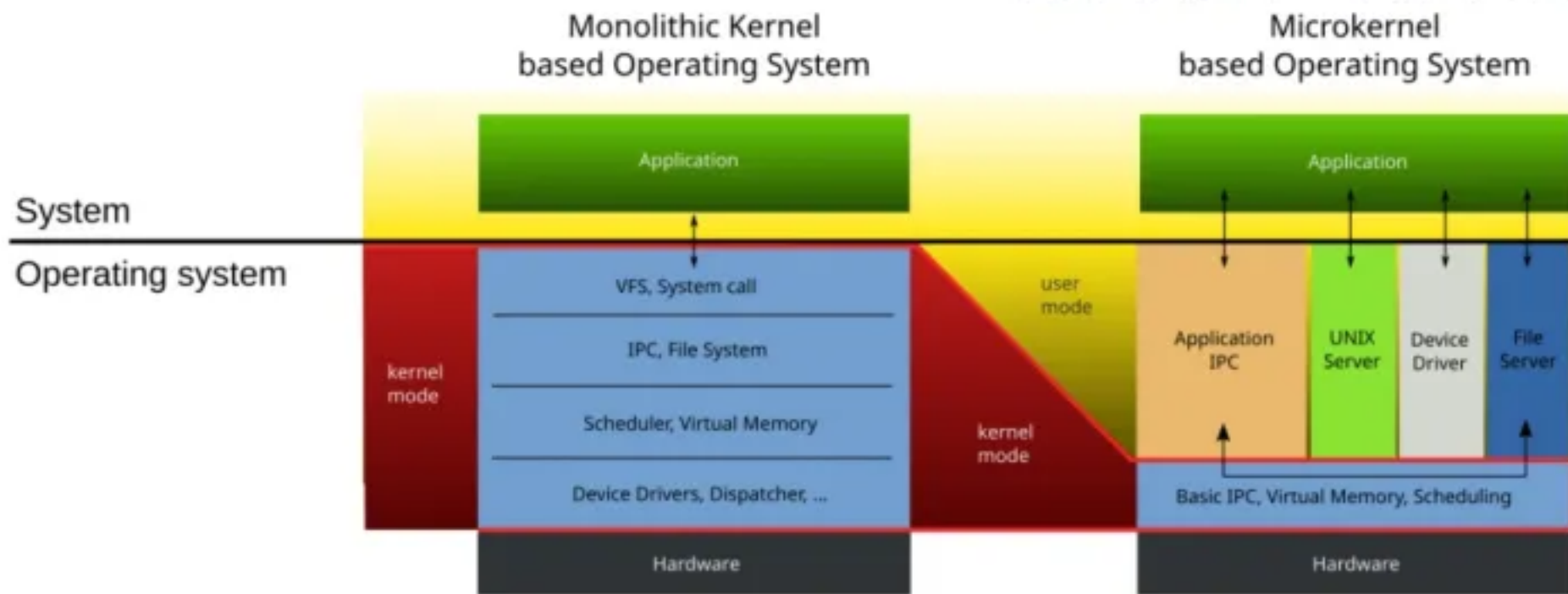


# 鸿蒙NEXT内核解读

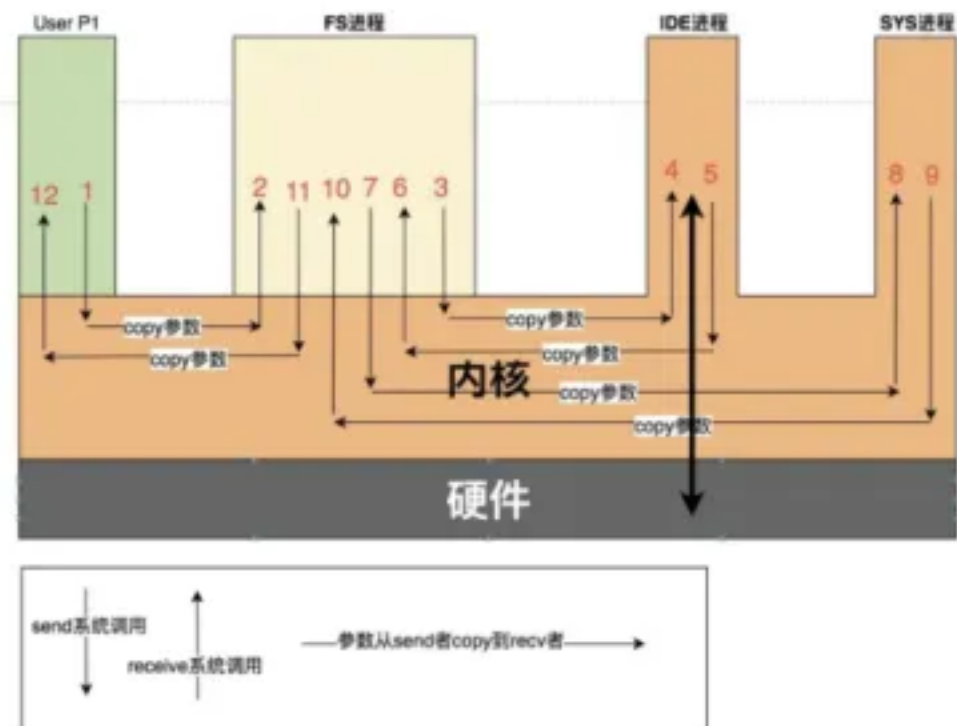
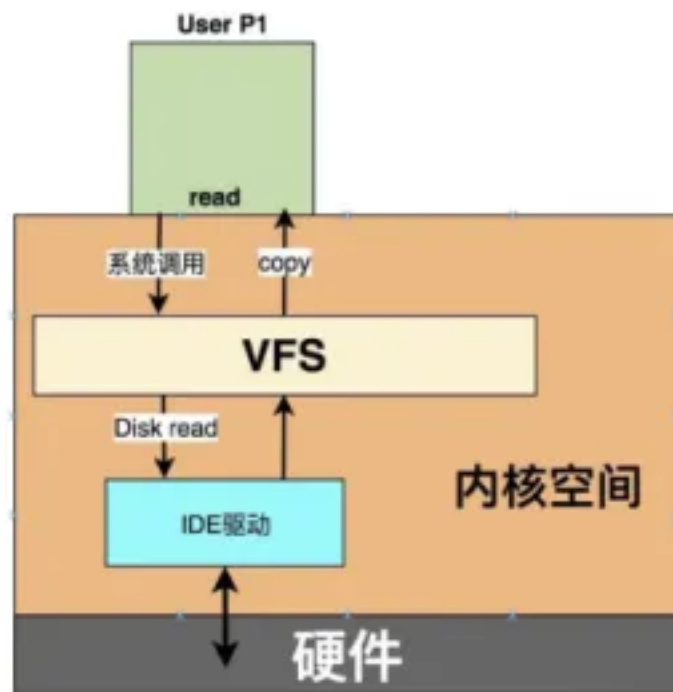
## ——微内核和宏内核概览



|     | 可靠性                                            | 安全                                                 | 执行效率                                           | 扩展性                                |
|-----|------------------------------------------------|----------------------------------------------------|------------------------------------------------|------------------------------------|
| 微内核 | 系统服务转移到用户空间，即使某个系统服务出现异常，也不会影响到内核崩溃，只是进程重启恢复服务 | 由于驱动程序 / 文件系统 / 系统服务在用户空间运行，与内核分离，故障或被攻击，不影响整系统    | 核心服务之间、文件操作、内存管理 page fault 中产生大量 IPC 通信，性能损耗大 | 移植性好，多场景灵活部署，汽车/手机/IOT             |
| 宏内核 | 内核服务异常，可能导致整个内核崩溃                              | 驱动和文件系统代码量为3000w行，占到总代码80%，过去4年1000个通用漏洞90%来自这部分代码 | 直接函数调用，性能损耗小                                   | 模块耦合严重，任何改动都可能影响到整个内核的稳定性，工作量大，风险大 |

# 鸿蒙NEXT内核解读

## ——微内核和宏内核IPC对比



宏内核通过函数调用访问特定的逻辑和数据。

微内核通过IPC(进程间通信)访问特定的逻辑和数据。

宏内核 1 个系统调用 read 的事，在微内核需要 12 次进程间通讯完成

函数调用到IPC的执行效率差距大

# 华为为什么需要新的内核？

## Linux内核不够用么？

- **通用但不够通用：**

Linux内核主要市场和其主要开发者在**服务器**和**云端**，针对 Android 手机这种交互与实时性高的场景，支持并不如意。尽管 Google 多年来，主导了诸如 mglru 新一代页面回收算法，提升内存使用效率，对 Android 设备同时提升保活和回收效率；process\_madvise 为 android 提供进程内存回收系统调用接口，供 Android appcompact 使用；Rust语言对Linux内核的支持；进程间通讯 binder；Cgroup 资源管理机制；针对 Linux 内核的改造多不胜数。但如

1. Linux 的抢占实时性补丁花费了 10 多年才部分合入。**Linux内核讲究通用而不是专用。**
2. Linux主要维护者代表云服务商的利益，不够重视嵌入式和电子设备领域。

- **定制化成本高**

- **宏内核的安全性弱点**

难以取得 ISO 26262 ASIL-D 的最高功能安全认证

- **华为有自主可控的战略需求**

独立于 Android / IOS，成为真正意义上的独立操作系统

狭义上：Android基于Linux内核构建的应用生态操作系统，而国内各家的 Origin OS / 澎湃 OS / FlymeOS / Color OS 是 Android 定制的操作系统。在用户端描述都叫操作系统，但技术层面差距异常巨大。

**Google 试图让原来无交互不实时服务器的内核，更适应 Android 移动设备。**



# 华为为什么需要新的内核？

## Linux内核不够用么？——汽车场景

### 汽车安全性问题

没有安全认证（如 ISO26262）不能上车使用

| 版本                      | 类型  | 安全认证等级 | 使用领域                    |
|-------------------------|-----|--------|-------------------------|
| 原生 Linux                | 宏内核 | 无      | 不能直接汽车上使用               |
| 华为鸿蒙                    | 微内核 | ASIL-D | 自动驾驶操作系统内核              |
| 中兴汽车GoldenOS            | 微内核 | ASIL-D | 覆盖智能车控、智能驾驶、智能座舱、智能网联场景 |
| 高通 QNX                  | 微内核 | ASIL-D | 智能座舱                    |
| 普华灵智ORIENTAIS操作系统使用领域   | 微内核 | ASIL-D | 动力、底盘等安全系统              |
| 赛福纳斯Safenux Linux内核产品   | 宏内核 | ASIL-B | 高阶辅助驾驶、智能座舱             |
| 高通-红帽（Red Hat）Linux操作系统 | 宏内核 | ASIL-B | 数字底盘                    |

微内核操作系统在汽车领域，有天然的优势，和强烈的需求。安全是汽车领域最大的需求

# 华为为什么需要新的内核？

## Linux内核不够用么？——手机场景

动效流畅

久用如新

滑动跟手

长用不热

多任务

启动迅速

.....

复杂繁多的手机用户体验需求

在手机移动场景，原生的 Linux 内核无法提供有力的支撑

cpu调度对前台应用及时响应

cpu调频需要符合 SoC 平台能效比模型

文件系统要抵抗碎片化

内存分配和回收需要高效和低延时

后台io请求和处理不能影响前台

高并发场景需要前台任务优先获取锁

手机需要精准测量和管理感知温度

Linux内核讲究通用而不是专用，所以手机大厂纷纷对内核动手术刀；而华为走向了国产操作系统完全取代之路。

任务计算需要考虑器件差异带来的负载差异

.....

Android 手机面对的用户体验问题，有定制化Linux内核的需求，然而维护、修改和测试成本高昂



华为拥有 IOT、车载、手机、平板、电视等各种产品线

## 为什么之前没有成熟的“通用”微内核？



- 性能损耗大：

微内核由于其设计，需要通过进程间通信（IPC）来实现进程之间的交互，这相较于宏内核的直接函数调用，会增加额外的性能开销。在早期，处理器硬件能力有限，速度是主要考虑因素，因此性能较低微内核并没有成为主流。

因此，微内核往往使用在特定的行业，特定的应用场景，主要强调安全性和稳定性，而非高性能。如工业控制、嵌入式系统领域。

手机属于无穷无尽的场景，器件多样性、应用多样性、内容多样性。对性能要求高，并发任务和复杂的计算需求。

华为想突破的是既要安全，又要稳定性，还要扩展性好，还可以普遍性能更好的全场景通用微内核操作系统

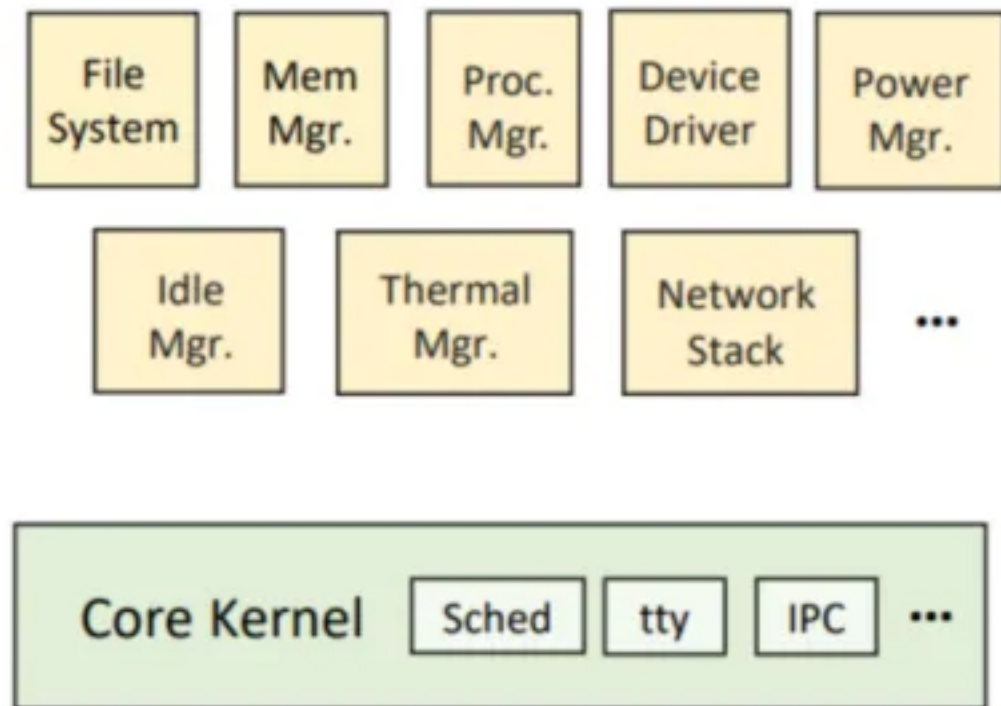


# 重新审视业界微内核问题

| 对比项 | 传统微内核   | 问题  | 鸿蒙内核 |
|-----|---------|-----|------|
| 最小性 | 最小化内核原则 | N/A | 保持   |
|     |         |     |      |
|     |         |     |      |
|     |         |     |      |

白兔 23040

# 保持最小性以保留微内核的优势



***HongMeng Kernel***

优势：

- 解耦、最小权限和良好隔离的操作系统服务
- 最小核心内核  
只包含调度器、进程间通信、访问控制、和必须的驱动程序，如 tty
- 细粒度访问控制



# 重新审视业界微内核问题

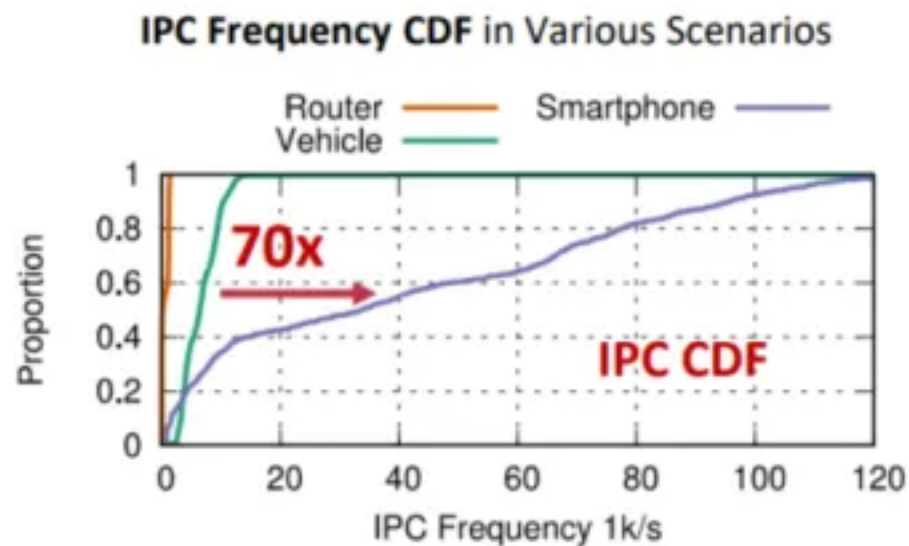
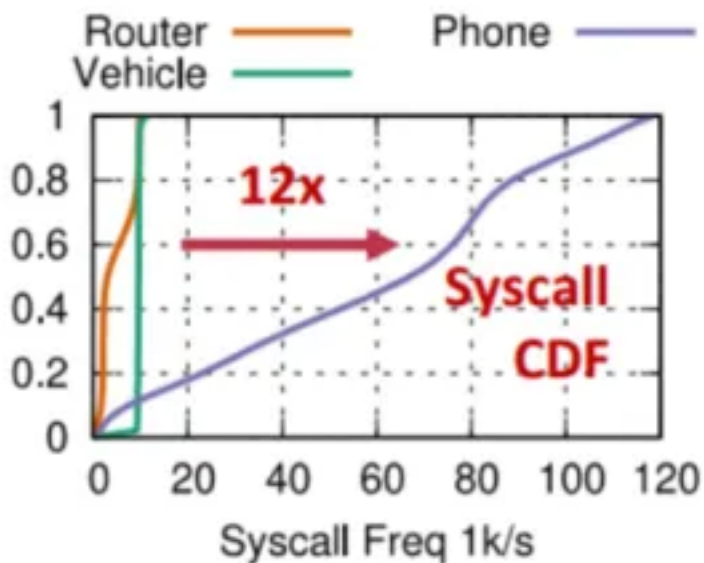
|        | 传统微内核     | 问题             | 鸿蒙内核  |
|--------|-----------|----------------|-------|
| 最小性    | 最小化内核原则   | N/A            | 保持    |
| IPC/隔离 | 所有服务在用户空间 | 手机场景超高的 IPC 频率 | 分层隔离类 |
|        |           |                |       |
|        |           |                |       |

白兔 3540

# 鸿蒙“通用”微内核所面临的挑战

## ——IPC

- 性能：IPC手机场景比专用设备路由器场景高 70 倍
  - 场景引起
  - 文件操作引起（IPC到FS）
  - 缺页异常引起
  - 场景的系统调用引起
- 页面缺页异常处理比 Linux 慢 3.4 倍

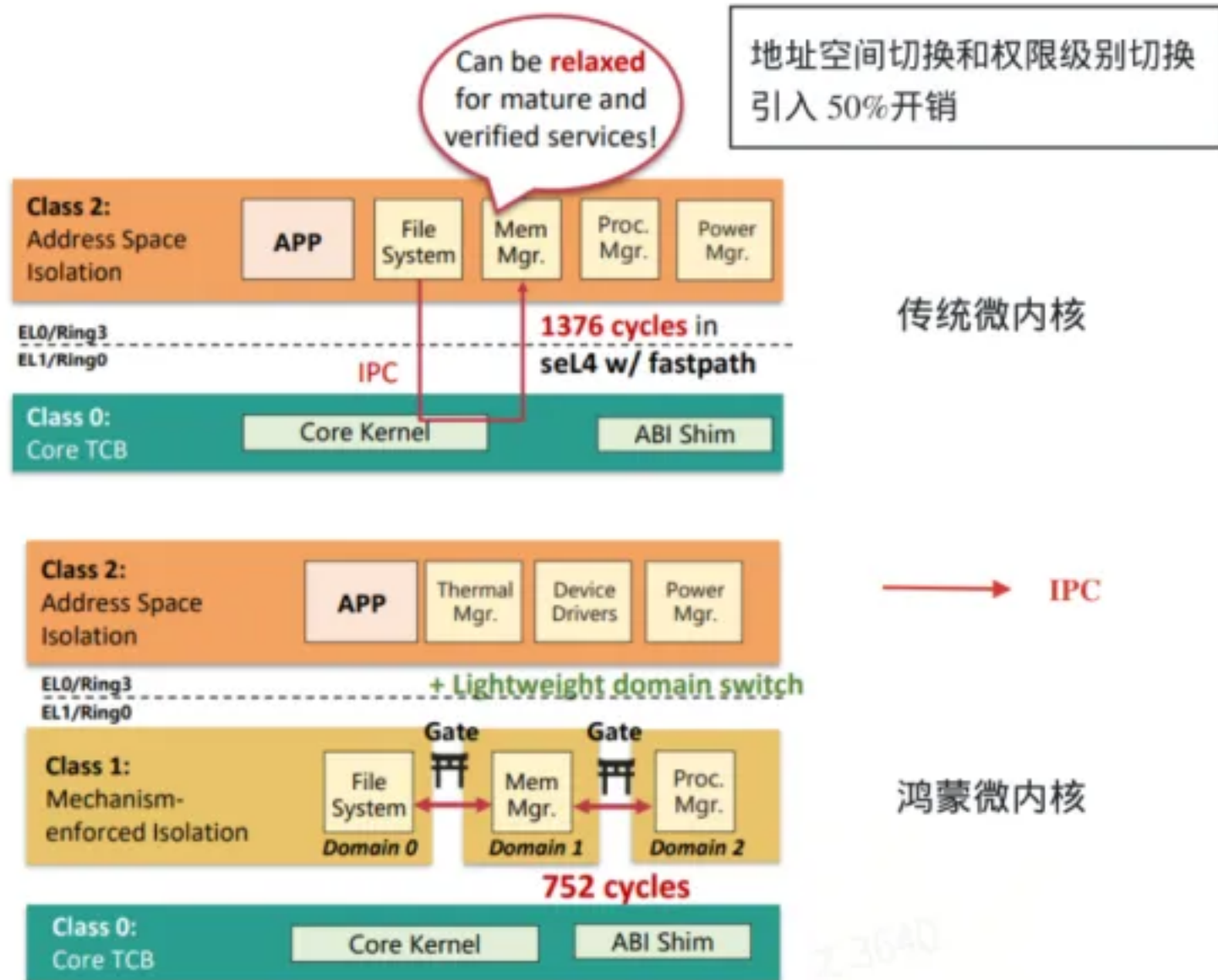


高 IPC 频率导致手机性能下降 2 到 3 倍

# 鸿蒙微内核分层隔离类

**分层隔离类**：HM在传统的「内核态」(IC0)和「用户态」(IC2)之间，多定义了一个特殊的隔离类IC1（也属于内核态），受信任的、对性能要求高的OS服务运行在这里，也就是说，通过放松受信任OS服务之间的隔离来减少IPC开销。

- IC0服务与内核之间不强制隔离，所有的IPC操作都是间接函数调用。）（跟当前宏内核一致，内核地址空间）
- IC1服务间的IPC通过最小化的上下文切换（切换指令和堆栈指针，不切换地址空间和调度），IPC延迟显著降低。IC1针对性能关键且已验证的OS服务，例如进程/内存管理服务。注意，这是一种权衡，对性能和安全的妥协折中，虽然IC1显著降低了IPC开销，但也引入了额外的攻击面，所以增加采用轻量级控制执行流完整性验证（CFI）和使用安全监视器来保护页面免受代码注入。是内核地址空间下的机制强制隔离区。
- IC2对非性能关键或包含第三方代码的服务（如Linux驱动程序）进行隔离。是用户地址空间。



Differentiated Isolation Classes in HongMeng

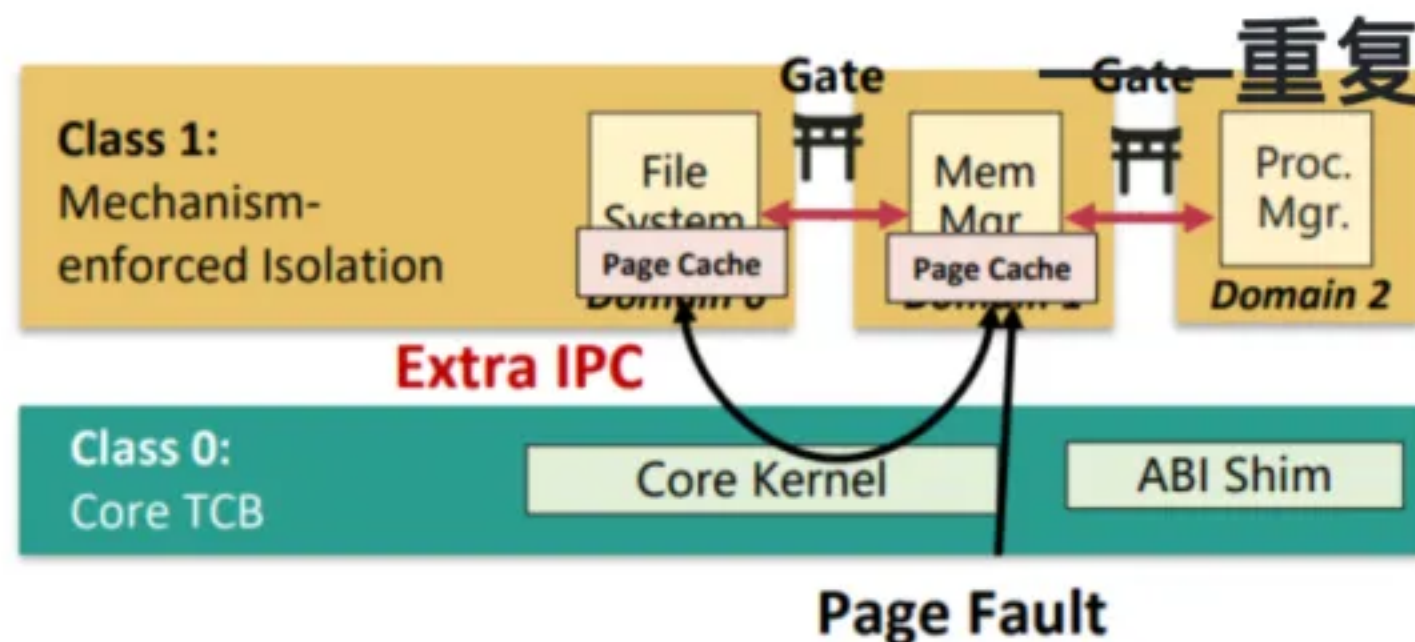


# 重新审视业界微内核问题

|        | 传统微内核     | 问题             | 鸿蒙内核   |
|--------|-----------|----------------|--------|
| 最小性    | 最小化内核原则   | N/A            | 保持     |
| IPC/隔离 | 所有服务在用户空间 | 手机场景超高的 IPC 频率 | 分层隔离类  |
| 服务分区管理 | 静态混合服务    | 状态重复记账         | 灵活组合服务 |
|        |           |                |        |

内部文件 3040

# 鸿蒙“通用”微内核所面临的挑战

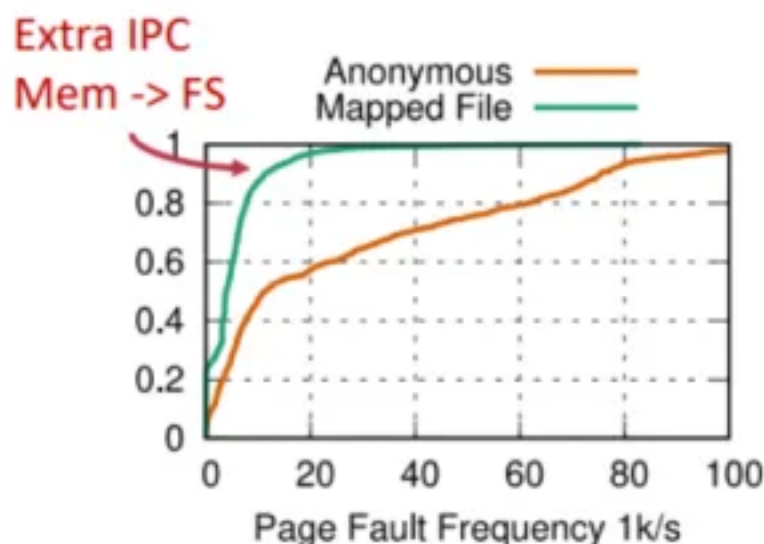


- 文件系统和内存管理两个服务需要密切协同工作

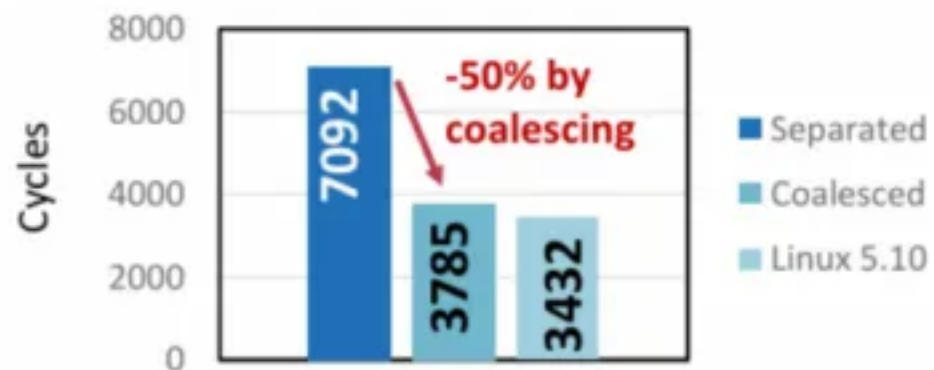
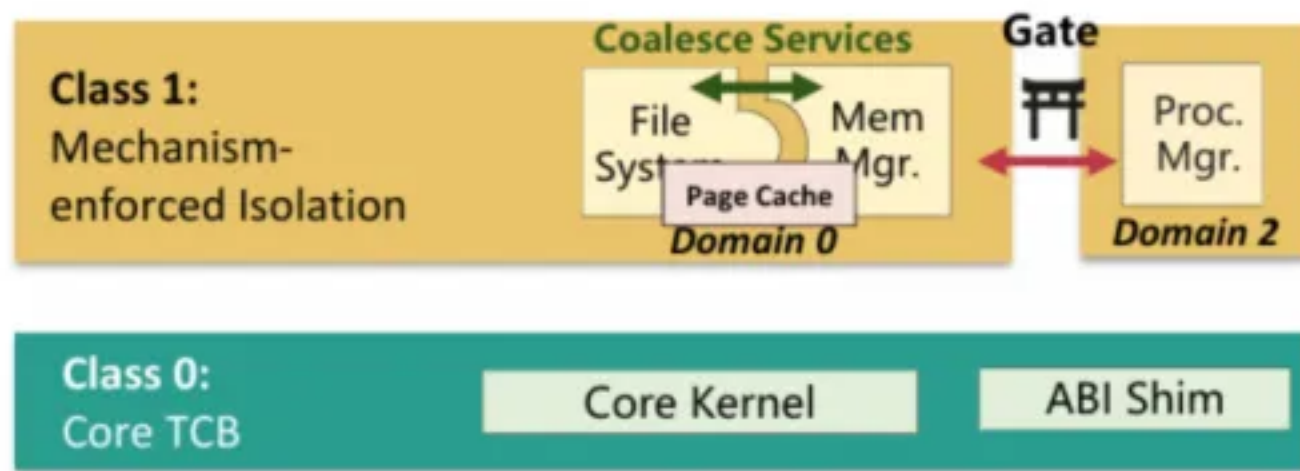
FS 和 Mem 之间大量存在文件缓存（page cache），他们之间会频繁做读 / 写 / 同步等操作，双重记账，会导致频繁 IPC 和显著的内存占用。

- 文件映射的缺页异常，要比匿名页的缺页异常更多

表明会导致额外的 Mem -> FS 的 IPC



# 鸿蒙微内核在性能关键场景中 合并耦合服务



*Page Fault Latency of mapped files*

- 合并耦合服务（IPC -> 函数调用）

1. 减少 IPC 频率
2. 消除重复记账

- 文件映射的缺页异常耗时恢复到 Linux 性能水准
- 服务的隔离与合并，可以灵活组装部署，仅在强调性能的手机场景合并

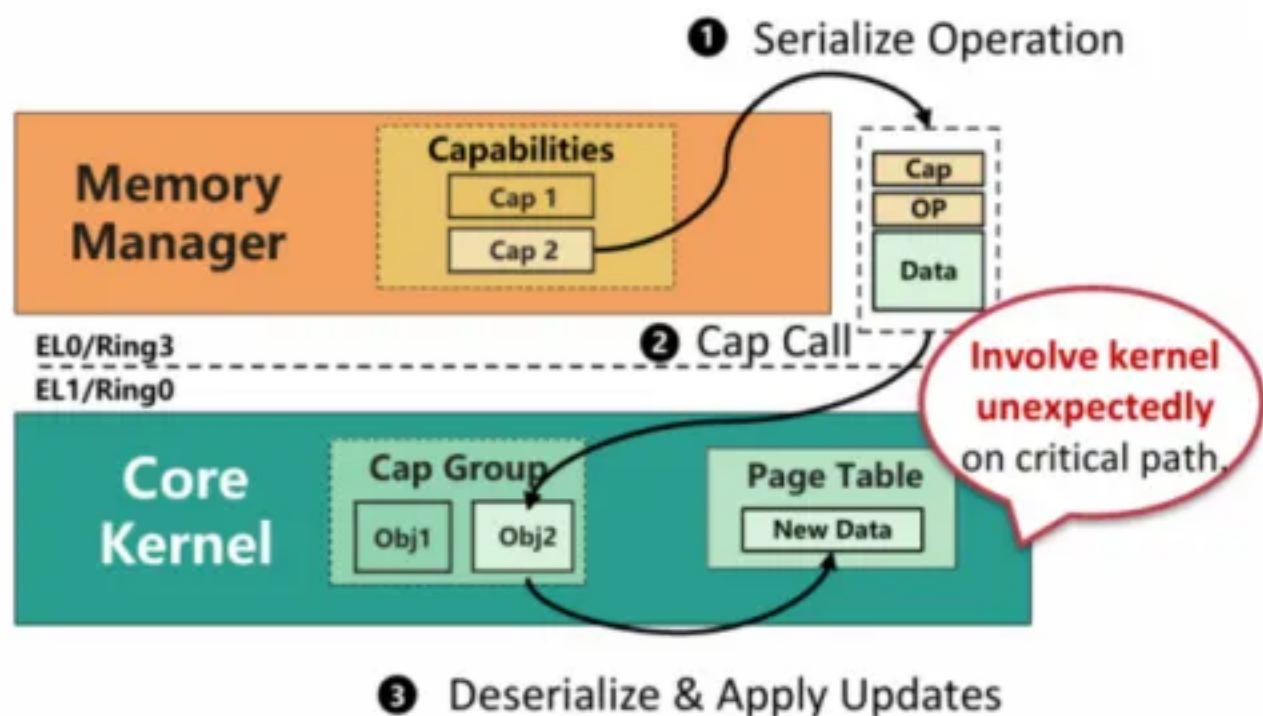


# 重新审视业界微内核问题

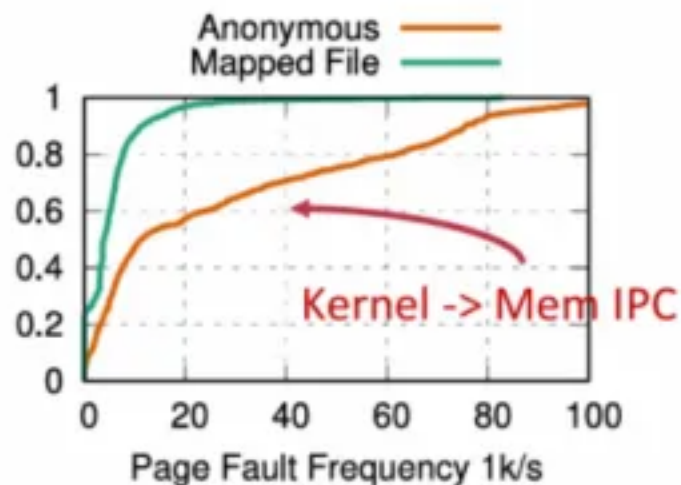
|        | 传统微内核     | 问题             | 鸿蒙内核        |
|--------|-----------|----------------|-------------|
| 最小性    | 最小化内核原则   | N/A            | 保持          |
| IPC/隔离 | 所有服务在用户空间 | 手机场景超高的 IPC 频率 | 分层隔离类       |
| 服务分区管理 | 静态混合服务    | 状态重复记账         | 灵活合并服务      |
| 访问控制   | 基于能力      | 隐藏内核对象         | 基于地址令牌的访问控制 |

# 鸿蒙“通用”微内核所面临的挑战

## 内核数据耗时

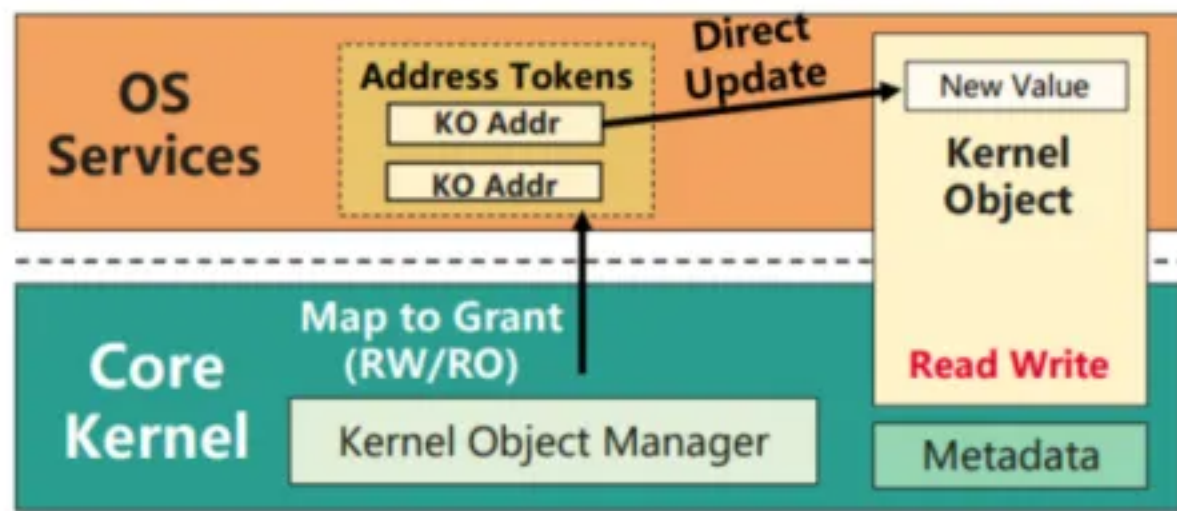
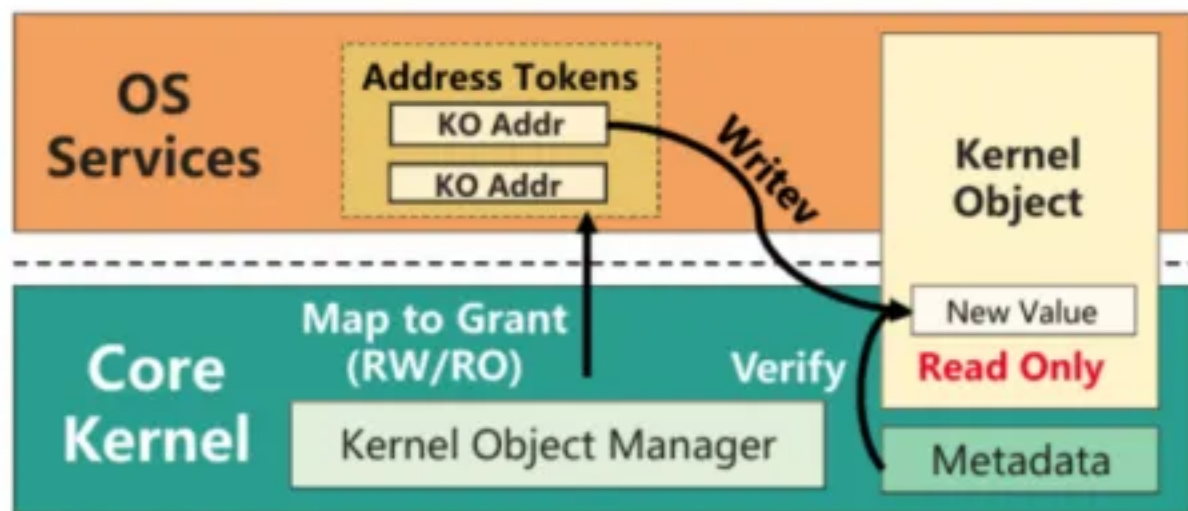


- 内核关键数据会被服务频繁更新  
内存服务和最小内核之间频繁 IPC，引入相当大的开销



白787-3640

# 鸿蒙微内核直接映射内核数据

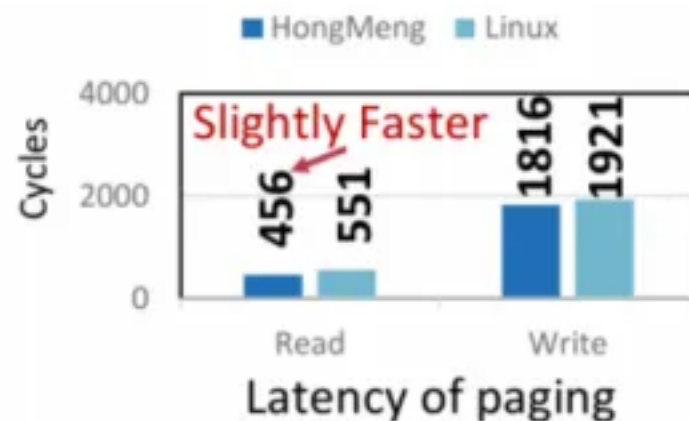
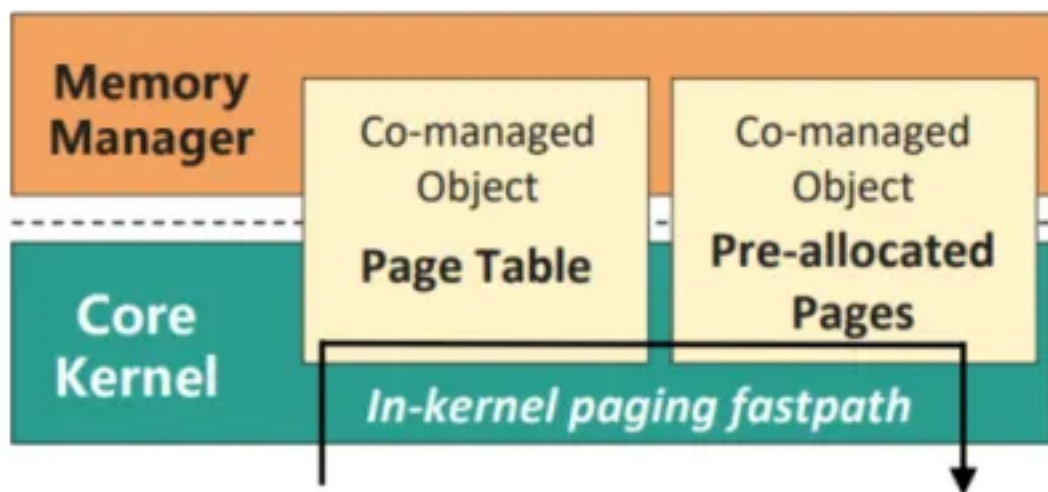


- 以地址为令牌标识，直接映射

1. 映射授予权限，取消映射撤销权限
2. 服务直接读 RO，使用特殊系统调用 writev 并在最小内核进行权限验证
3. 一旦授予写 RW，无须经过最小内核
4. 风险：一旦地址令牌授权服务，内核无法监控后续操作



# 鸿蒙微内核地址令牌实现高效的对象共同管理的对象共同管理



- **问题点：用户空间分页缓慢**

1. 应用程序访问未与物理页面建立映射关系虚拟地址 -> 最小内核触发缺页异常中断 -> IPC -> Mem进程检查地址并分配新的页面，然后返回最小内核更新页表，最后回到应用程序。

- **方案：地址令牌实现高效的对象共同管理**

包括页表、预分配页

白话讲 3540

# 重新审视业界微内核问题

|        | 传统微内核     | 问题             | 鸿蒙内核          |
|--------|-----------|----------------|---------------|
| 最小性    | 最小化内核原则   | N/A            | 保持            |
| IPC/隔离 | 所有服务在用户空间 | 手机场景超高的 IPC 频率 | 分层隔离类         |
| 服务分区管理 | 静态混合服务    | 状态重复记账         | 灵活合并服务        |
| 访问控制   | 基于能力      | 隐藏内核对象         | 基于地址令牌的访问控制   |
| 接口     | POSIX     | 需求大于 POSIX     | Linux ABI 兼容层 |

# 为什么仅仅是 POSIX 不能兼容生态

- **POSIX (Portable Operating System Interface)** , 可移植操作系统接口 , 保证应用程序在**源码层次**的可移植性 , API 兼容。(未编译)

POSIX 可选实现 , 不能保证对应的操作系统有实现

约定 libc 库、系统调用、命令行工具等 , 统一 API 的函数名、返回值、参数类型等

- **ABI (Application Binary Interface)** , 应用程序二进制接口 , 跟芯片架构和系统绑定 , 保证应用程序在**可执行文件层次**的可移植性。(已编译)

数据类型的大小、布局和对齐

函数调用过程 , 参数如何使用 cpu 通用寄存器、如何返回上一个函数等

二进制可执行文件格式 (ELF)

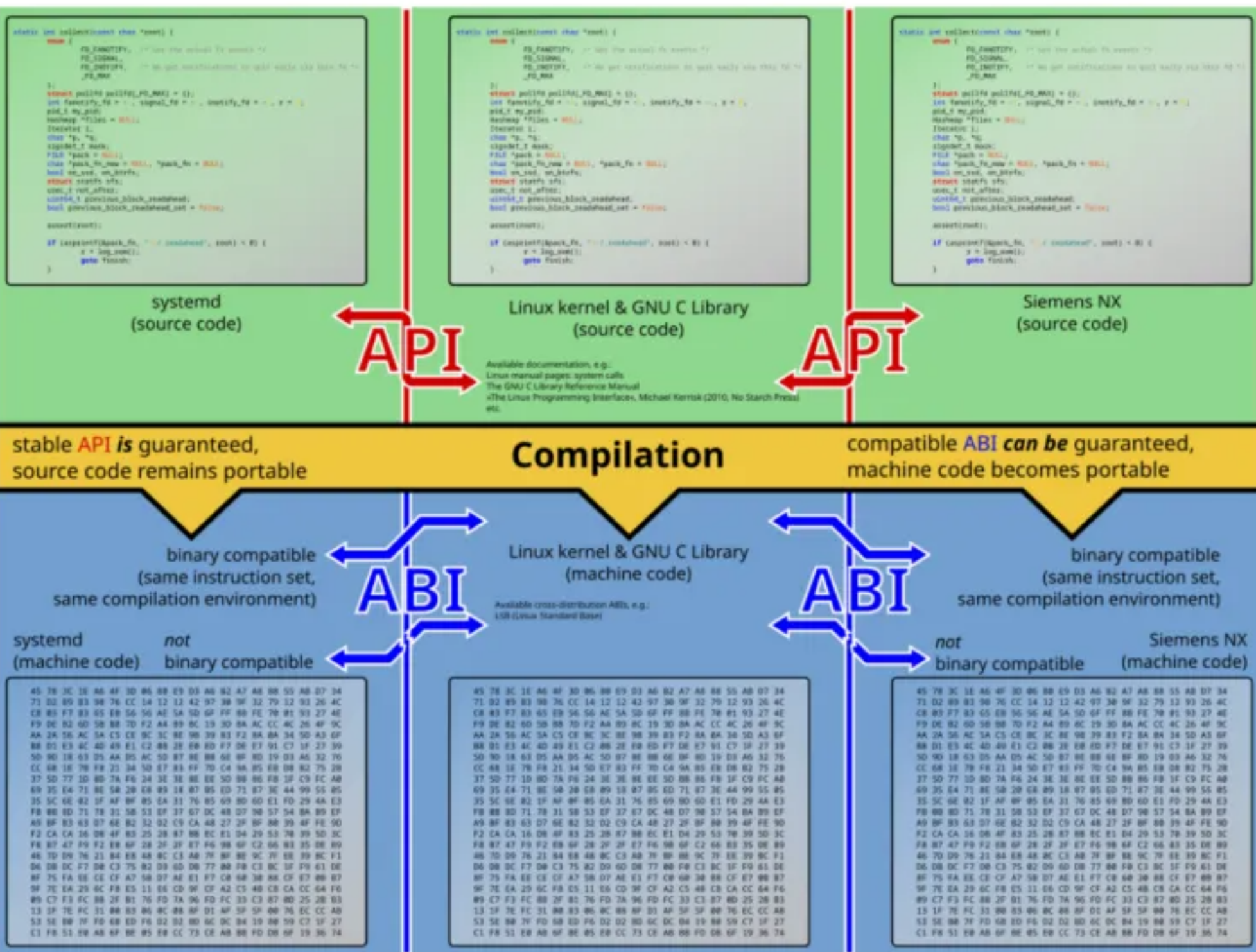
栈帧布局、虚拟地址空间、动态链接和共享库等

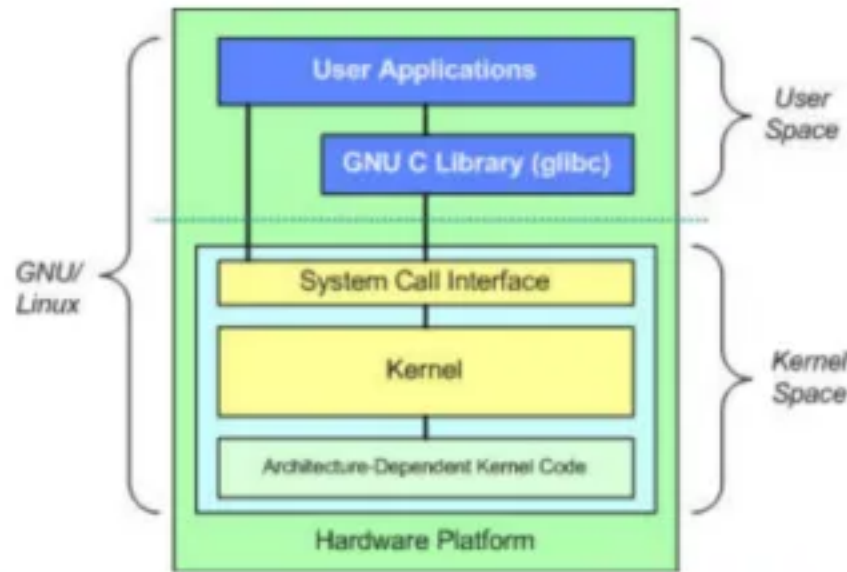
保证 ABI 稳定 , 还需要适配 sysfs/procfs 虚拟文件系统——用户和内核空间交互接口

ABI 是机器语言约定的 API , 描述机器代码如何访问数据结构与运算程序

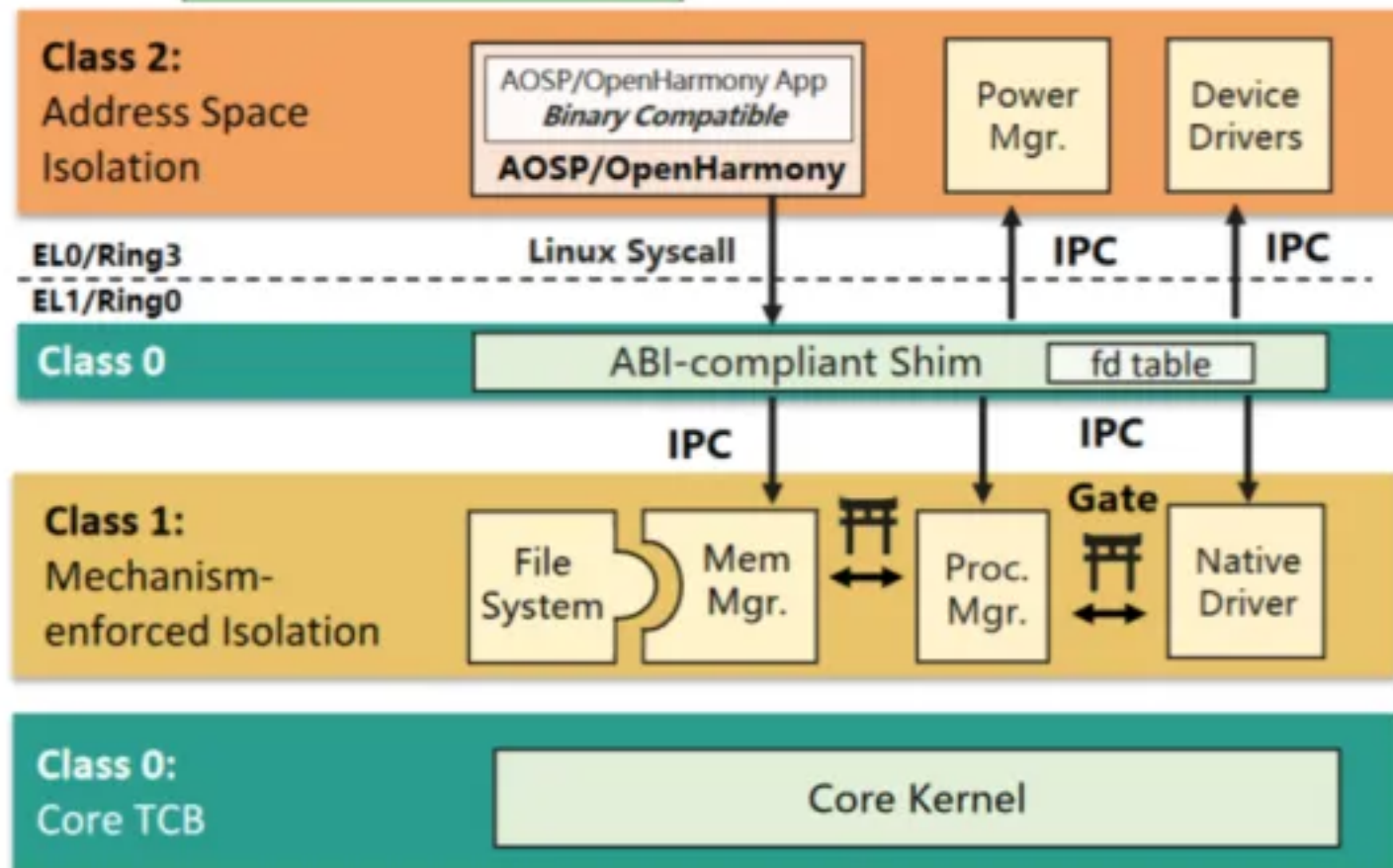
至此 , 上层 Android 层 和 开源鸿蒙应用 , 都可以换掉 Linux 内核 , 成功运行在鸿蒙内核上







## 通过符合 ABI 标准的中间层 实现 Linux 二进制兼容



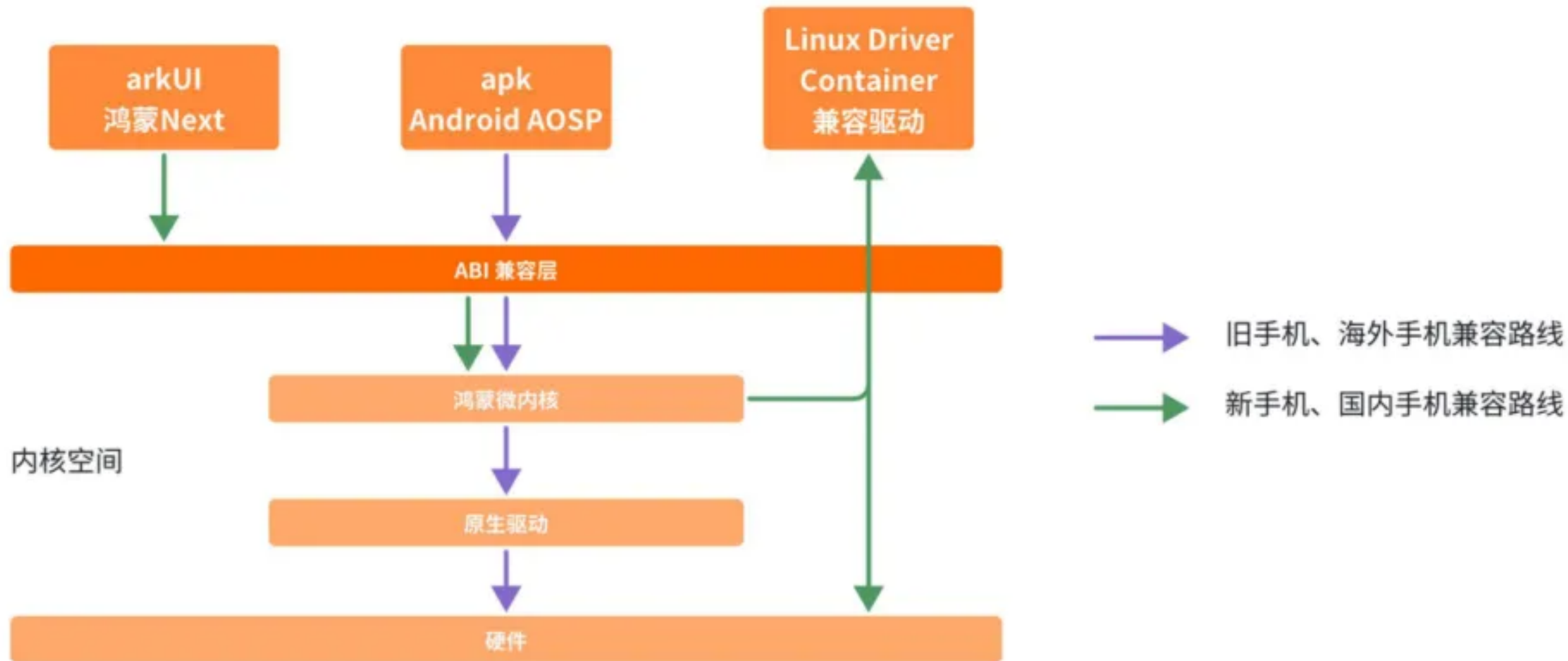
- ABI兼容层，由原来函数调用，变为重定向系统调用到IPC到系统服务
- 管理全局状态的中央存储库  
高效实现如 fd 轮询
- 支持替换上层 AOSP / OpenHarmony
- 支持替换底层 Linux 内核 / 鸿蒙微内核

至此，上层 Android 层和开源鸿蒙应用，都可以换掉 Linux 内核，成功运行在鸿蒙内核上



# 鸿蒙Next 兼容路线图

用户空间



只要保证 ABI 兼容层稳定，可以做到保证生态兼容，和切换微内核

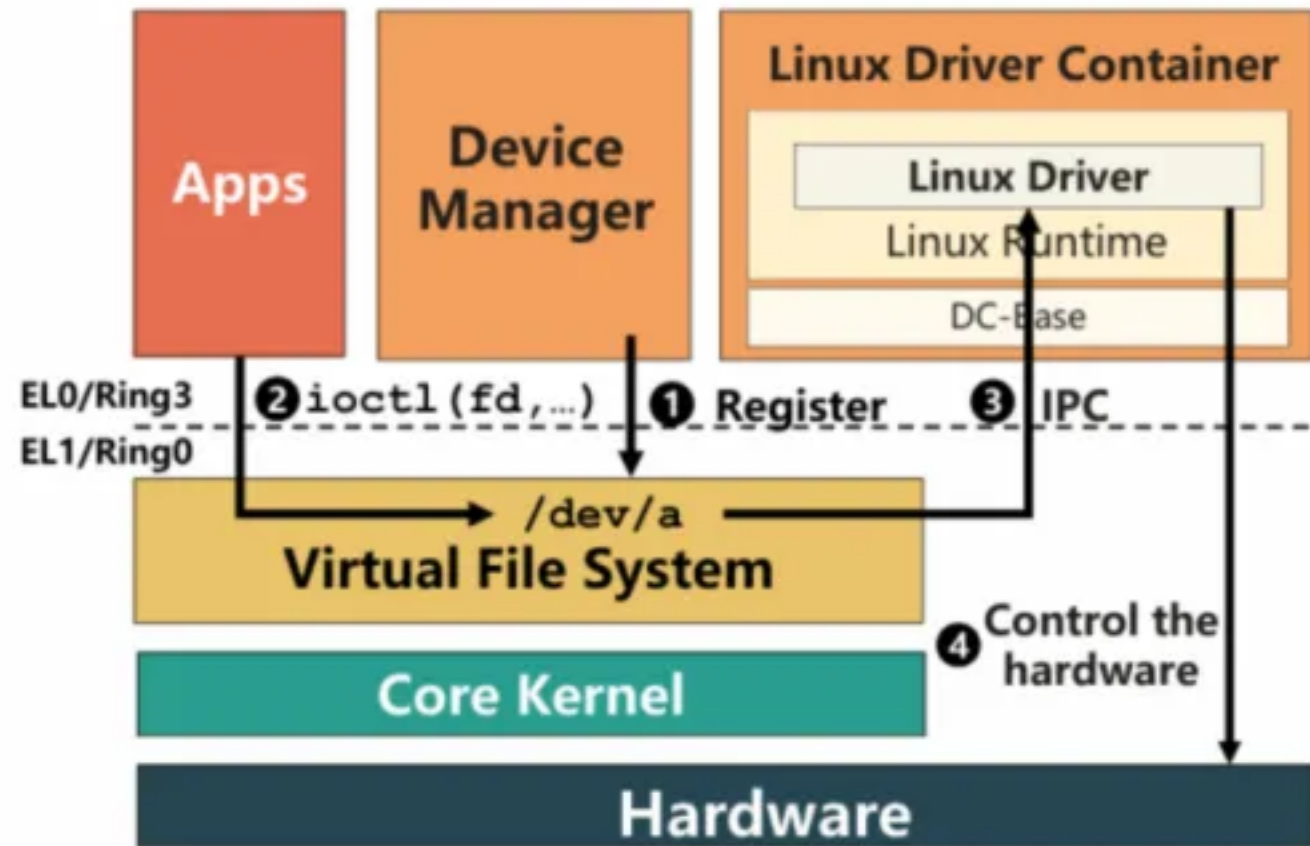


# 重新审视业界微内核问题

|        | 传统微内核      | 问题/需求          | 鸿蒙内核          |
|--------|------------|----------------|---------------|
| 最小性    | 最小化内核原则    | N/A            | 保持            |
| IPC/隔离 | 所有服务在用户空间  | 手机场景超高的 IPC 频率 | 分层隔离类         |
| 服务分区管理 | 静态混合服务     | 状态重复记账         | 灵活合并服务        |
| 访问控制   | 基于能力       | 隐藏内核对象         | 基于地址令牌的访问控制   |
| 接口     | POSIX      | 需求大于 POSIX     | Linux ABI 兼容层 |
| 驱动程序   | 虚拟机 / 移植驱动 | 高度可重用          | 驱动容器          |

817523040

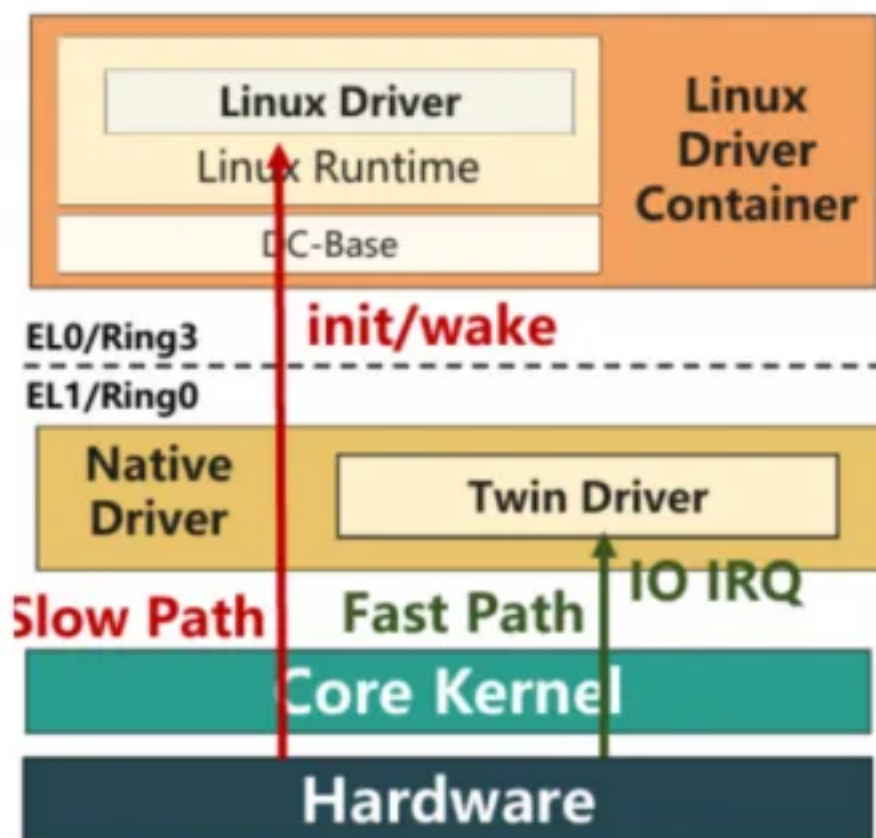
# 鸿蒙微内核使用驱动容器重用 驱动程序



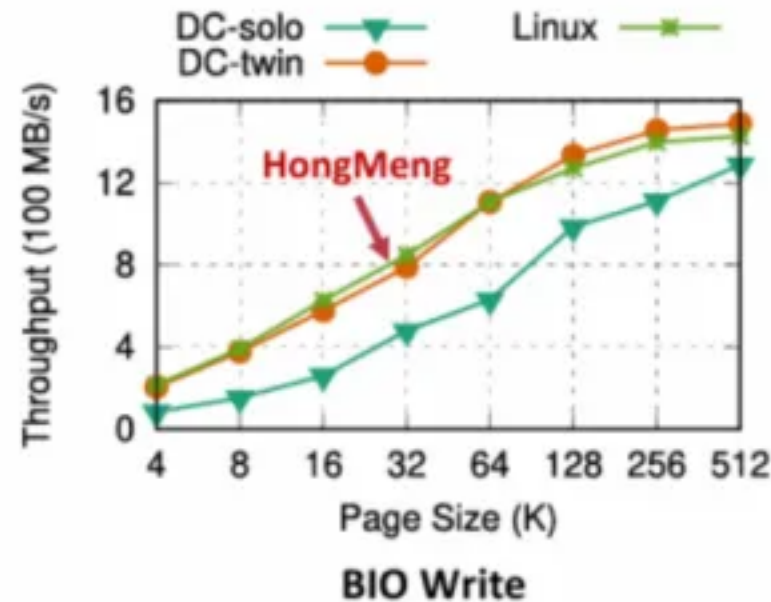
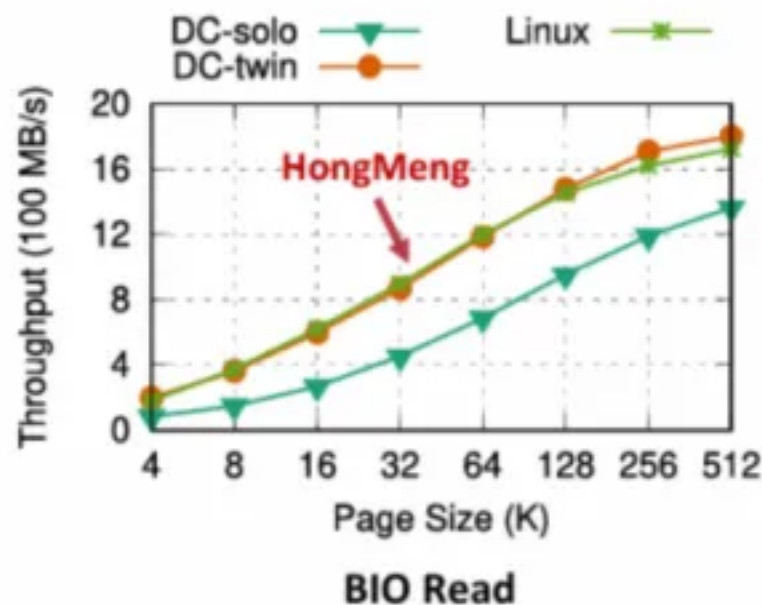
*Driver Containers in HongMeng*

- 在用户空间运行一个最小 Linux 容器来重用驱动程序
- DC-base驱动容器重定向必要的内核API。如 Kthread, Kernel Memory API
- Linux 容器升级需要改动很小（从 4.19升级到 5.10，只改动不到 100处）

# 鸿蒙微内核使用数据/控制分离提升关键驱动性能



Driver Containers in HongMeng



- 对性能关键驱动，如UFS驱动，创建 twin 驱动初始化/休眠/唤醒等操作，保留在容器驱动里；Twin 驱动处理频繁数据读写

UFS 驱动使用数据 / 控制分离后，UFS 读写性能甚至优于 Linux 原生水平



# 鸿蒙微内核的部署



*OS Kernel for  
Routers/Switches*



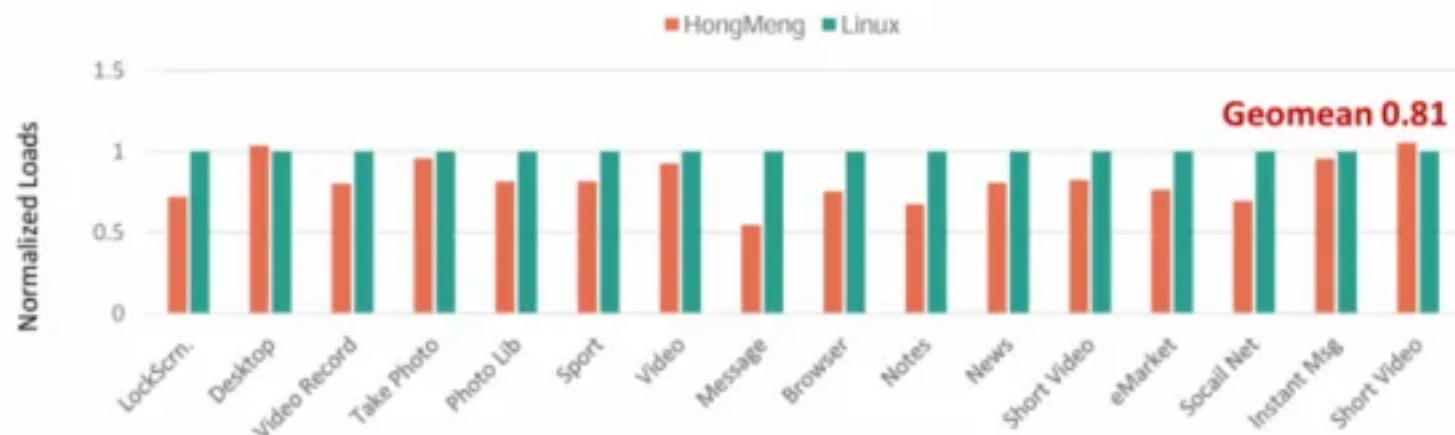
*OS Kernel for  
Smart Vehicles*



*OS Kernel for  
Smartphones/Tablets*

- 部署在数千万台设备中
- 相同的代码库，不同的配置，可应用于不同场景设备上
- 通过最高级别安全认证 **CC-EAL6+ (security)** 和 **ASIL-D (safety)**

# 鸿蒙微内核的性能对比



- 网络提升
- 上下文切换提升
- 内存 / 文件读写 持平
- Fork/Clone 衰退（通过并发弥补）
- 整机负载低 19%
- 连续应用启动快 17%，丢帧率降低 10%

| Benchmark Commands <sup>1</sup> | Unit | Linux | HM    | Norm. <sup>2</sup> |
|---------------------------------|------|-------|-------|--------------------|
| lat_unix -P 1                   | µs   | 10.23 | 10.39 | 0.98               |
| lat_tcp -m 1K                   | µs   | 21.22 | 17.19 | 1.23               |
| lat_tcp -m 16K                  | µs   | 24.54 | 18.9  | 1.29               |
| lat_tcp -m 1K (Same Core)       | µs   | 21.21 | 17.19 | 1.23               |
| lat_tcp -m 1K (Cross core)      | µs   | 37.96 | 25.66 | 1.47               |
| lat_udp -m 1K                   | µs   | 17.83 | 19.48 | 0.91               |
| lat_udp -m 16K                  | µs   | 23.63 | 22.02 | 1.07               |
| lat_udp -m 1K (Same Core)       | µs   | 18.04 | 19.55 | 0.92               |
| lat_udp -m 1K (Cross core)      | µs   | 34.17 | 26.84 | 1.27               |
| bw_tcp -m 16K                   | MB/s | 1812  | 3109  | 1.71               |
| bw_unix                         | MB/s | 7124  | 8478  | 1.19               |
| bw_mem 256m bcopy               | MB/s | 17696 | 17202 | 1.02               |
| bw_mem 512m ffd                 | MB/s | 14514 | 14593 | 0.99               |
| bw_mem 256m tcp                 | MB/s | 17492 | 15867 | 0.91               |
| bw_mem 512m bw                  | MB/s | 34771 | 35318 | 1.01               |
| bw_file 16 512m seqonly         | MB/s | 8976  | 9396  | 1.04               |
| bw_mmap_rd 512M mmap_only       | MB/s | 26073 | 27520 | 1.05               |
| lat_mmap 512m                   | µs   | 3315  | 3628  | 0.91               |
| lat_pagefault                   | µs   | 0.83  | 0.78  | 1.06               |
| lat_ctx -s 16 8                 | µs   | 4.53  | 3.41  | 1.32               |
| bw_pipe                         | MB/s | 3808  | 4127  | 1.08               |
| lat_pipe                        | µs   | 9.00  | 7.88  | 1.14               |
| lat_proc exec                   | µs   | 336   | 1305  | 0.26               |
| lat_proc fork                   | µs   | 323   | 1280  | 0.25               |
| lat_proc shell                  | µs   | 2269  | 4778  | 0.47               |
| lat_clone (create thread)       | µs   | 28.6  | 54.3  | 0.52               |

network

Avg. +21%

mem + file

Similar

Context Switches

Fork

Clone

+32%

-75%

-48%

极客网 3540

# 鸿蒙全栈自主可控

## 全面突破操作系统核心技术

鸿蒙

盘古大模型 / MindSpore框架

方舟多媒体引擎

方舟图形引擎

星盾安全架构 / 分布式安全 / ...

DevEco Studio / DevEco Testing / ...

ArkUI / ArkUI-X

方舟编译器 / 毕昇编译器

ArkTS / 仓颉编程语言

ArkData分布式智能数据底座

分布式软总线 / 通途协议 / 星闪 / ...

EROFS / HMDFS / ...

鸿蒙内核

AI

多媒体

图形

安全隐私

集成开发环境

编程框架

编译器

编程语言

数据库

全场景互联

文件系统

OS内核

GPT / Gemini / TensorFlow / PyTorch / ...

Dolby Vision / Dolby Atmos / ...

OpenGL ES / Skia / Metal / ...

SEP / Keystore / Keychain / MAC / ...

Xcode / Android Studio / VS Code / ...

SwiftUI / Jetpack Compose / Flutter / ...

GCC / CL / Clang+LLVM / ...

Object-C / Swift / Java / Kotlin / ...

Core Data / SwiftData / Room / ...

WiFi / Bluetooth / ...

EXT / FAT / NTFS / APFS / SquashFS / ...

Linux / Unix / ...

其他

值 什么值得买

华为因为有自主可控的战略需求，操作系统相关领域，上到应用编程语言，下到芯片制造工艺，全面突破



# 鸿蒙NEXT内核解读

解读数据来源于：

上海交通大学陈海波和华为中央软件研究所于 2024 OSDI ( Operating Systems Design and Implementation 操作系统领域顶级学术会议 )

论文《Microkernel Goes General: Performance and Compatibility in the HongMeng Production Microkernel》

白居文 20240